BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

As rescanning documents *will not* correct images,
Please do not report the images to the
Image Problem Mailbox.

IEEE Std 100-1996

The IEEE Standard Dictionary of Electrical and Electronics Terms

Sixth Edition

Standards Coordinating Committee 10, Terms and Definitions
Jane Radatz, Chair

This standard is one of a number of information technology dictionaries being developed by standards organizations accredited by the American National Standards Institute. This dictionary was developed under the sponsorship of voluntary standards organizations, using a consensus-based process.

ISBN 1-55937-833-6

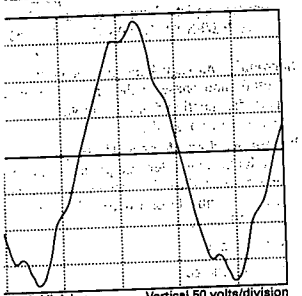
nction of the input signal amplitude.
cy multiplier or a frequency divider is a
ionic conversion transducer. *See also:* het-
transducer; transducer.
(ED) 161-1971w

(1) (data transmission) Nonlinear distor-
ransducer characterized by the appearance
rmonics other than the fundamental com-
put wave is sinusoidal. *Note:* Subharmonic
occur.
(PE) 599-1985w

cal area networks) A form of interference
ration of signals according to the relation-
is an integer greater than one and *f* is the
equency.
(C/LM) 802.7-1989

le wave input, output components at fre-
n integer multiple of the applied sine wave
(IM) 1057-1994

ical representation of the distortion of the
m. (See figure below.) *See also:* distortion
onic distortion.



distortion example

\/PE/PSPD/T&D) 1100-1992, 1250-1995, C62.48-1995

stortion that appears as harmonics of a single-
(PE) 1143-1994

The ratio of the root-sum-square (rss) value
onics to the root-mean-square (rms) value of

(for voltage) $= \dfrac{\sqrt{E_2^2 + E_3^2 + E_4^2 \cdots}}{E_1}$

(for current) $= \dfrac{\sqrt{I_2^2 + I_3^2 + I_4^2 \cdots}}{I_1}$

(IA) 519-1992

ge power (TR and pre-TR tubes) The total
y power transmitted through the fired tube in
equencies other than the fundamental frequen-
by the transmitter.
(ED) 161-1971w

haracteristic Harmonics that are not produced
tor converter equipment in the course of nor-
These may be a result of beat frequencies; a
of characteristic harmonics and the fundamen-
alance in the ac power system, asymmetrical
r cycloconverter operation.
(IA) 519-1992

aint relay A restraint relay so constructed that
s restrained by harmonic components of one or
input quantities.
(PE/SWG) C37.100-1992

harmonic components.

s A series in which each component has a
t is an integral multiple of a fundamental fre-
(SP) [32]

hone ringer A telephone ringer that responds
nating current within a very narrow frequency
A number of such ringers, each responding to a
quency, are used in one type of selective ringing.
(EEC/PE) [119]

harmonic test (rotating machinery) A test to determine directly the value of one or more harmonics of the waveform of a quantity associated with a machine, relative to the fundamental of that quantity. *See also:* asynchronous machine.
(PE) [9]

harmonization The process of ensuring that profiles do not overlap or conflict.
(C/PA) 14252-1996

harness A component with a design of straps that is fastened about the worker in a manner so as to contain the torso and distribute the fall arrest forces over at least the upper thighs, pelvis, chest, and shoulders with means for attaching it to other components and subsystems.
(NESC/PE/T&D) 1307-1996, C2-1997

harsh environment (nuclear power generating station) An environment expected as a result of the postulated service conditions appropriate for the design basis and post-design basis accidents of the station. (A design basis accident is that subset of a design basis event which requires safety function performance). Harsh environments are the result of a loss of cooling accident (LOCA)/high energy line break (HELB) inside containment and post-LOCA or HELB outside containment.
(PE) 323-1974s

hartley A unit of information content, equal to one decadal decision, or the designation of one of ten possible and equally likely values or states of anything used to store or convey information. *Notes:* 1. A hartley may be conveyed by one decadal code element. One hartley equals (log of 10 to base 2) times one bits. 2. If, in the definition of information content, the logarithm is taken to the base ten, the result will be expressed in hartleys. *Synonym:* dit. *See also:* bit.
(IT/PE) [123], 599-1985w

Hartley oscillator An electron tube or solid state circuit in which the parallel-tuned tank circuit is connected between grid and plate, the inductive element of the tank having an intermediate tap at cathode potential, and the necessary feedback voltage obtained across the grid-cathode portion of the inductor. *See also:* radio-frequency generator.
(IA) 54-1955w

Harvard class architecture A computer architecture with separate paths to main storage for instructions and data, allowing for a high memory bandwidth. *Contrast:* Von Neumann architecture.
(C) 610.10-1994

hash To calculate the hash value for a given item. *See also:* hashing.
(C) 610.5-1990

hash address *See:* hash value.

hash addressing *See:* hashing.

hash clash *See:* collision.

hash coding *See:* hashing.

hash function In hashing, the function used to determine the position of a given item in a set of items. *Note:* The function operates on a selected field, called a key, in each item and the function is generally a many-to-one mapping. *Synonyms:* calc algorithm; key transformation function. *See also:* algebraic coding function; digit transformation function; division transformation function; key folding function; key transformation; mid-square function; multiplication transformation function; radix transformation function.
(C) 610.5-1990

hash index *See:* hash value.

hashing A technique for arranging a set of items, in which a hash function is applied to the key of each item to determine its hash value. The hash value identifies each item's primary position in a hash table, and if this position is already occupied, the item is inserted either in an overflow table or in another available position in the table. *Synonyms:* hash coding; randomizing; scatter storage. *See also:* collision resolution; hash addressing; open-address hashing; separate chaining.
(C) 610.5-1990

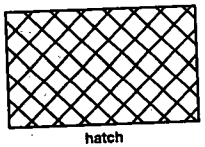
hash search The use of a hash function and collision resolution to locate an item in a hash table.
(C) 610.5-1990

hash table A two-dimensional table of items in which a hash function is applied to the key of each item to determine its hash value. The hash value identifies each item's primary position in the table, and if this position is already occupied, the item is inserted either in an overflow table or in another available position in the table.
(C) 610.5-1990

hash total The result of summing two or more values of a set for purposes of validation or error detection. *Synonym:* control total.
(C) 610.5-1990

hash value The number generated by a hash function to indicate the position of a given item in a hash table. *Synonyms:* hash address; hash index.
(C) 610.5-1990

hatch A series of one or more sets of evenly spaced parallel lines within a closed boundary on a display surface. *See also:* crosshatch.



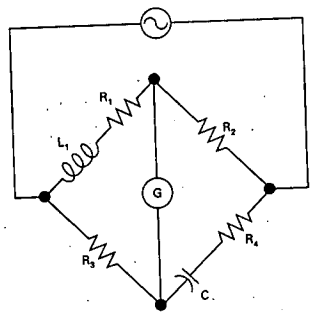
hatch

(C) 610.6-1991

HATS *See:* head and torso simulator.

hauptnutzzeit *See:* utilization time.

Hay bridge A 4-arm alternating-current bridge in which the arms adjacent to the unknown impedance are nonreactive resistors and the opposite arm comprises a capacitor in series with a resistor. *Note:* Normally used for the measurement of inductance in terms of capacitance, resistance, and frequency. Usually, the bridge is balanced by adjustment of the resistor that is in series with the capacitor, and of one of the nonreactive arms. The balance depends upon the frequency. It differs from the Maxwell bridge in that in the arm opposite the inductor, the capacitor is in series with the resistor. *See also:* bridge.



Hay bridge

(EEC/PE) [119]

hazard (1) (nuclear power generating station) A specified result of a design basis event that could cause unacceptable damage to systems or components important to safety.
(PE) 384-1981s

(2) (overhead power lines) A threat to the health, survival, or reproduction of an organism from some natural or artificial agent or event.
(PE/T&D) 539-1990

(3) An intrinsic property or condition that has the potential to cause harm or damage.
(DEI) 1221-1993

hazard beacon (illuminating engineering) An aeronautical beacon used to designate a danger to air navigation. *Synonym:* obstruction beacon.
(EEC/IE) [126]

hazard current (1) (health care facilities) For a given set of connections in an isolated power system, the total current that would flow through a low impedance if it were connected between either isolated conductor and ground. The various hazard currents are: fault hazard current; monitor hazard current; total hazard current. *See also:* fault hazard current; monitor hazard current; total hazard current.
(EMB) [47]

(2) (health care facilities) For a given set of connections in an isolated system, the total current that would flow through a low impedance if it were connected between either isolated conductor and ground. Fault Hazard Current: The hazard current of a given isolated system with all devices connected except the line isolation monitor. Monitor Hazard Current: The hazard current of the line isolation monitor alone. Total Hazard Current: The hazard current of a given isolated system with all devices, including the line isolation monitor, connected.
(NEC/NESC) [86]

hazard-free logic A group of logic circuits that are not subject to failures due to logic failure conditions.
(C) 610.10-1994

hazardous (classified) locations Class I Locations. Class I locations are those in which flammable gases or vapors are or may be present in the air in quantities sufficient to produce explosive or ignitible mixtures. Class I locations shall include those specified in (a) and (b) below.

a) Class I, Division 1. A Class I, Division 1 location is a location:
   1) in which hazardous concentrations of flammable gases or vapors exist continuously intermittently, or periodically under normal operating conditions; or
   2) in which hazardous concentrations of such gases or vapors may exist frequently because of repair or maintenance operations or because of leakage; or
   3) in which breakdown or faulty operations of equipment or processes might release hazardous concentrations of flammable gases or vapors, and might also cause simultaneous failure of electric equipment. This classification usually includes locations where volatile flammable liquids or liquefied flammable gases are transferred from one container to another; interiors of spray booths and areas in the vicinity of spraying and painting operations where volatile flammable solvents are used; locations containing open tanks or vats of volatile flammable liquids; drying rooms or compartments for the evaporation of flammable solvents; portions of cleaning and dyeing plants where hazardous liquids are used; gas generator rooms and other portions of gas manufacturing plants where flammable gas may escape; inadequately ventilated pump rooms for flammable gas or for volatile flammable liquids; the interiors of refrigerators and freezers in which volatile flammable materials are stored in open, lightly stoppered, or easily ruptured containers; and all other locations where hazardous concentrations of flammable vapors or gases are likely to occur in the course of normal operations.

b) Class I, Division 2. A Class I, Division 3 location is a location
   1) in which volatile flammable liquids or flammable gases are handled, processed, or used, but in which the hazardous liquids, vapors, or gases will normally be confined within closed containers or closed systems from which they can escape only in case of accidental rupture or breakdown of such containers or systems, or in case of abnormal operation of equipment; or
   2) in which hazardous concentrations of gases or vapors are normally prevented by positive mechanical ventilation, and which might become hazardous though failure or abnormal operation of the ventilating equipment; or
   3) that is adjacent to a Class I, Division I location, and which hazardous concentrations of gases or vapors might occasionally be communicated unless such communication is prevented by adequate positive-pressure ventilation from a source of clean air, and effective guards against ventilation failure are provided.

Osborne **McGraw-Hill**
2600 Tenth Street
Berkeley, California 94710
U.S.A.

For information on translations and book distributors outside of the U.S.A., please write to Osborne **McGraw-Hill** at the above address.

A complete list of trademarks appears on page 760.

C: The Complete Reference

1234567890 DODO 8987

ISBN 0-07-881263-1

Jeffrey Pepper, Acquisitions Editor
Kris Jamsa, Technical Reviewer
Lyn Cordell, Project Editor
Judy Wohlfrom, Text Design
Bay Graphics Design Associates, Cover Design

pointer array. Often, as in the spreadsheet example, an easy method exists to index the pointer array and link it with the sparse-array elements. This makes accessing the elements of the sparse array nearly as fast as it would be if it were a normal array. The linked-list version is very slow by comparison because it must use a linear search to locate each element. Even if extra information were added to the linked list to allow faster accessing, it would still be slower than the pointer array's direct access. The binary tree certainly speeds up the search time, but when compared with the pointer array's direct-indexing capability, it still seems sluggish. If the hashing algorithm is properly chosen, the hashing method can often beat the binary tree in access times, but it will never be faster than the pointer-array approach.

When choosing an approach, the rule of thumb is to use the pointer-array implementation when possible—it is the fastest in terms of access time. If memory is in short supply, then you have no choice but to use the linked-list or binary-tree approach.

Choosing an Approach

When deciding whether to use a linked list, a binary tree, a pointer array, or hashing to implement a sparse array, consider two factors: speed and memory efficiency.

When the array is very sparse, the most memory-efficient approaches are the linked list and the binary tree because only array elements that are actually in use have memory allocated to them. The links themselves require very little additional memory and usually have a negligible effect. The pointer array requires that the entire pointer array exits even if some of its elements are not used. This means that not only must the entire pointer array fit in memory, but that enough memory must be left over for the application to use. This could be a serious problem in certain applications, but it may not be a problem at all in others. Usually you can decide if the pointer array causes a problem for you by calculating the approximate amount of free memory and determining whether that is sufficient for your program. The hashing method lies somewhere in the middle, between the pointer-array approach and the linked-list or binary-tree approach. Although the hashing method does require the existence of the entire physical array (even if all of it is not used), it may be that the physical array is still smaller than a pointer array (which needs at least one pointer for each logical-array location).

However, when the array is fairly full, the situation changes. In this case the pointer array makes better use of memory. The reason is that the tree and linked-list implementations need two pointers, whereas the pointer array has only one pointer. For example, if a 1000-element array was full and pointers were 2 bytes long, then both the binary tree and linked list would use 4000 bytes for pointers. The pointer array, on the other hand, would need only 2000—a savings of 2000 bytes. In the hashing method even more memory is "wasted" to support the array.

By far the fastest approach, in terms of execution, is the

```
char *cell_name;
{
  int h,loc;

  /* produce the hash value */
  loc=*cell_name-'A';
  loc+=(atoi(&cell_name[1])-1) * 26; /* WIDTH columns * num rows */
  h=loc/10;

  /* return the value if found */
  if(hash[h].index==loc)  return(hash[h].val);
  else { /* try next location */
    while(h<MAX) {  /* find a free loc */
      h=hash[h].next;
      if(h==-1) break; /* not found */
      if(hash[h].index==loc) return(hash[h].val);
    }
    printf("not in array\n");
    return -1;
  }
}
```

Analysis of Hashing

In its best case, which occurs rarely, each physical index created by the hash is unique, and access times approximate that of direct indexing. This means that no hash chains are created and all look-ups are essentially direct accesses. However, this is seldom the case because it requires that the logical indexes be evenly distributed throughout the logical-index space. In the worst case, which is also rare, a hashed scheme degenerates into a linked list. This can happen when the hashed values of the logical indexes are all the same. In the average and most likely case, the hash method can access any specific element in the same time that it would take to use a direct index divided by some constant that is proportional to the average length of the hash chains. In using hashing to support a sparse array, it is critical that the hashing algorithm spread the physical index evenly so that long hash chains are avoided. Also, hashing is best applied to situations in which you know that there is a limit to the number of array locations actually required.

pointed to by the hashed value is occupied, then it searches for the first free location. When a free location is found, the value of the logical index as well as the value of the array element are stored. It is necessary to store the logical index because it will be needed when that element is searched for.

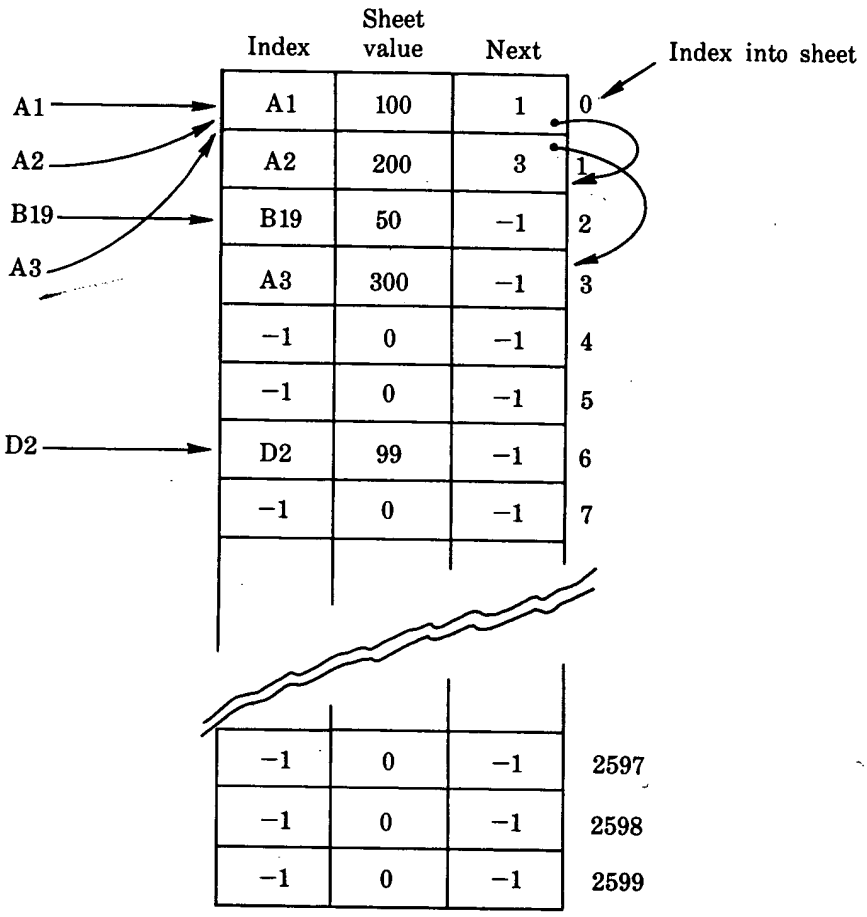
```
/* compute hash and store value */
void store(cell_name,v)
char *cell_name;
int v;
{
  int h, prior,loc;

  /* produce the hash value */
  loc=*cell_name-'A';
  loc+=(atoi(&cell_name[1])-1) * 26;/* WIDTH columns * num rows */
  h=loc/10;

  /* store in the location unless full or
     store there if logical indexes agree - i.e., update.
  */
  if(hash[h].index==-1 || hash[h].index==loc) {
    hash[h].index=loc;
    hash[h].val=v;
  }
  else { /* try next location */
    while(h<MAX) {  /* find a free loc */
      prior=h;
      h++;
      if(hash[h].index==-1) break;
    }
    if(h==MAX) {
      printf("hash error or array full\n");
      return;
    }
    hash[h].val=v;
    hash[h].index=loc;
    hash[prior].next=h; /* add the link */
  }
}
```

To find the value of an element already in the array, you must first compute its physical address. Then check to see if the logical index stored in the physical array matches that of the index of the logical array that is requested. If it does, then that value is returned; otherwise, the chain is followed. The function **find()**, which does this, is shown here.

```
/* compute hash and return value */
int find(cell_name)
```



Assume that the value of A1 is 100, A2 is 200, A3 is 300, B19 is 50, and D2 is 99.

Figure 20-3. A hashing example: how the logical index maps onto the physical array

The procedure **store()** converts a cell name into a hashed index into the array **hash**. Notice that if the location directly

location—whether the pointers are pointing to actual information or not. This may be a serious limitation for certain applications, but in general it is not a problem.

Hashing

Hashing is the process of extracting the index of an array element directly from the information that will be stored there. The index generated is called the *hash*. Traditionally, hashing has been applied to disk files as a means of decreasing access time. However, the same general methods can be used as a means of implementing sparse arrays. The procedure used with the preceding example of a pointer array used a special form of hashing called *direct indexing*, in which each key maps onto one and only one array location. That is, each hashed index is unique. (Note, however, that the pointer-array approach does not require a direct-indexing hash—it was just an obvious approach to the spreadsheet problem.) However, in actual practice, such direct-hashing schemes are few; a more flexible method is required. In this section you will see how hashing can be generalized to allow greater power and flexibility.

If you think about the spreadsheet example, it is clear that even in the most rigorous environments not every cell in the sheet will be used. For the sake of this example, assume no more than 10 percent of the potential locations in almost all cases are occupied by actual entries. This means that if the spreadsheet has the dimensions 26 by 100 (2,600 locations), then only 260 will ever be used at any one time. Therefore, the size of the largest array necessary to hold all the entries is 260 elements. The problem then becomes this: How do the logical-array locations get mapped onto and accessed from this smaller physical array? The answer is the use of a *hash chain*.

When the user of the spreadsheet (the logical array) enters a formula for a cell, the cell location (which is defined by its name) is used to produce an index (a hash) into the smaller physical array. Assume that the physical array is called **sheet**. The index is derived from the cell name by converting the name into a number, as shown in the example of the pointer array. However,

this number is then divided by 10 to produce an initial entry point into the array. (Remember that in this example the physical array is only 10 percent as big as the logical array.) If the location that is referenced by this index is free, then the logical index and the value are stored there. Otherwise, the array **sheet** is searched for an open element. When an unused element is found, the information is placed there and a pointer to this location is stored in the original element. This situation is depicted in Figure 20-3.

To find an element in the physical array given the logical-array index, you first transform the logic index into its hash value. Then check the physical array at the index generated by the hash to see if the logical index stored there matches the one that you are searching for. If it does, then return the information. Otherwise, follow the hash chain until either the proper index is found or the end of the chain is reached.

To see how this procedure is actually applied to the spreadsheet program, you must define the following array of structures, which acts as the physical array:

```
#define MAX 260

struct htype {
  int index; /* actual index */
  int val;  /* actual value of the array element */
  int next; /* index of next value with same hash */
} hash[MAX];
```

Before this array can be used, it must be initialized. To indicate an empty element, the following function initializes the **index** field to −1, a value that by definition cannot be generated. The −1 in the **next** field is used to indicate the end of a hash chain.

```
/* init the hash array */
void init()
{
  register int i;

  for (i=0; i<MAX; i++) {
    hash[i].index=-1;
    hash[i].next=-1;  /* null chain */
    hash[i].val=0;
  }
}
```